A comparative study of search and optimization algorithms for the automatic control of physically realistic 2-D animated figures

A Comparative Study of Search and Optimization Algorithms for the Automatic Control of Physically Realistic 2-D Animated Figures

Alex Fukunaga[1]
Harvard University

Jon Christensen
Harvard University

J. Thomas Ngo[2]
Harvard University

Joe Marks[3]
Digital Equipment Corp.

[1]Current affiliation: Computer Science Dept., University of California, Los Angeles, CA.
[2]Current affiliation: Interval Research Corp., Palo Alto, CA.
[3]Current affiliation: Mitsubishi Electric Research Labs, Cambridge, MA.

Abstract

In the Spacetime Constraints paradigm of animation, the animator specifies what a character should do, and the details of the motion are generated automatically by the computer. Ngo and Marks [11, 12] recently proposed a technique of automatic motion synthesis that uses a massively parallel genetic algorithm to search a space of motion controllers that generate physically realistic motions for 2D articulated figures. In this paper, we describe an empirical study of evolutionary computation algorithms and standard function optimization algorithms that were implemented in lieu of the massively parallel GA in order to find a substantially more efficient search algorithm that would be viable on serial workstations. We discovered that simple search algorithms based on the evolutionary programming paradigm were most efficient in searching the space of motion controllers. [1]

[1] Portions of this report have been presented previously [7, 8].

1 Introduction

Computer animation has traditionally been a tedious and time-consuming process. In most current computer animation systems, the animator is given minimal assistance from the computer. For example, key-framing is the most common method for generating animations [6]; in computer-assisted key-frame animation the animator specifies a character's configuration at key points in time, and the computer interpolates the intermediate configurations between the key frames. This method suffers from two major problems:

- the level of automation is minimal, and
- it is difficult to achieve physically realistic (and therefore visually plausible) animation with interpolation methods.

Automatic motion synthesis for articulated figures is the problem posed by the Space-time Constraints (SC) paradigm for animation proposed by Witkin and Kass [17].

In the SC paradigm, the animator specifies:

- the physical structure of the character;
- the actuators that control the character's internal configuration; and
- criteria for evaluating the character's motion.

The computer is expected to automatically generate a physically realistic and realizable trajectory for the character that is near optimal[2] with respect to the criteria given by the animator.

SC problems have two properties which make them difficult to solve:

- The solution space is multimodal — the number of locally optimal trajectories a character can follow is exponential, and many are far from the global optimum. In addition, there may be many dissimilar solutions that are near the global optimum.
- The search space is discontinuous — small changes in the behavior of a character (i.e., the operation of its actuators) can result in large changes to the trajectory it follows.

Early motion-synthesis algorithms avoid the difficulties of global optimization by settling for some form of local optimization: an initial trajectory (provided by the animator) is subjected to perturbation-based local optimization [17, 1, 2]. However, local optimization left the animator with the difficult task of supplying a reasonable initial trajectory.

Recently, a new approach to the motion-synthesis problem was proposed [11, 12, 15]. In this approach, the goal is not to compute the figure's trajectory directly, but instead to generate a *motion controller* that, when executed in a simulated physical environment, will produce the desired motion. The key aspects of any particular embodiment of this approach are:

[2] This problem is NP-hard, so guaranteed optimality is too difficult a requirement.

- how the motion controller is represented; and
- how the space of possible controllers is searched.

Ngo and Marks [11, 12] proposed an algorithm in which the controller is represented as a bank of mutually independent stimulus-response (SR) rules; we shall refer to this kind of controller as a banked stimulus-response (BSR) controller. Information from the physical environment is used to determine which rule is active at any given time in the physical simulation. The space of possible BSR controllers is searched using a massively parallel genetic algorithm.

While this approach is successful, a major shortcoming is the expense of the search algorithm: finding a simple motion controller for a five-rod articulated figure took 30-60 minutes on a 4096 processor CM-2 Connection Machine. If the approach is to be of practical value, a more efficient search algorithm is required.

The goal of our research is to explore search algorithms that would be viable on workstation-class serial machines. We present a study of optimization algorithms for searching the space of BSR controllers, comparing the various algorithms empirically on several SC problems.

2 The Motion Synthesis Algorithm

This section briefly describes the BSR controller representation and massively parallel GA used by Ngo and Marks in their motion synthesis algorithm [11, 12].

2.1 The BSR Controller Representation

A BSR controller governs a vector $\vec{\theta}(t)$ of joint angles, given information about the physical environment in the form of a vector $\vec{S}(t)$ of *sense variables*. Sample sense variables for an articulated figure are listed in Table 1.

$\theta_1, \theta_2, \ldots, \theta_{n-1}$	Joint angles
$f_1, f_2, \ldots, f_{n+1}$	Contact forces at rod endpoints
y_{cm}	Height of center of mass
$\dot{y}_{\mathrm{cm}}$	Vertical velocity of center of mass

Table 1: Components of the vector $\vec{S}$ of sense variables for an n-rod articulated figure.

The controller contains N stimulus-response rules. Every rule i is specified by stimulus parameters $\vec{S}^{\mathrm{lo}}[i]$ and $\vec{S}^{\mathrm{hi}}[i]$, and response parameters $\vec{\theta}^{0}[i]$ and $\tau[i]$. Based on the instantaneous value of the sense vector $\vec{S}(t)$, exactly one rule is active at any one time. In particular, each rule i receives a score based on how far the instantaneous sense vector $\vec{S}(t)$ falls within the hyperrectangle whose corners are $\vec{S}^{\mathrm{lo}}[i]$ and $\vec{S}^{\mathrm{hi}}[i]$. The highest-scoring rule

Set i_{active} to 1
for $t = 1$ **to** $t_{\max}$
  Cause joint angles $\vec{\theta}(t)$ to approach $\vec{\theta}^0[i_{\text{active}}]$
    with time constant $\tau[i_{\text{active}}]$
  Simulate motion for time interval t
  Measure sense variables $\vec{S}(t)$
  Possibly change i_{active}, based on $\vec{S}(t)$
end for
Assign the controller a *fitness value* based on
  how well the simulated motion meets the
  animator-supplied task criteria

Fig. 1. Pseudocode for a BSR controller.

is said to be marked *active*. (If $\vec{S}(t)$ is not inside the hyperrectangle associated with any rule, the rule active in the previous time step remains active.) The joint angles $\vec{\theta}(t)$ are made to approach the target values $\vec{\theta}^0[i_{\text{active}}]$ prescribed by the active rule i_{active}. Figure 1 summarizes how a BSR controller behaves and is evaluated.

2.2 The Massively Parallel Genetic Algorithm

Ngo and Marks' original motion synthesis algorithm used a massively parallel GA to search the space of possible BSR controllers. In this algorithm, shown in Figure 2, each candidate solution, or *genome*, was assigned to a single processor, and each generation of genomes was evaluated in parallel. The details of the initial randomization and mate selection, crossover, and mutation, which are specific to this application, are described elsewhere [11, 12].

The evaluation function used to measure the fitness of a candidate solution was specific to each class of SC problem. For example, in an instance of a SC problem in which the objective was to generate an articulated figure that jumped as high as possible, the evaluation function could be a function of the highest altitude achieved by the figure's center of mass.

Although it first seemed that the match between the genetic algorithm and SIMD massive parallelism was ideal, the issue of suitability is more complex. Ngo and Marks observed some incompatibilities between the CM-2 architecture and the motion synthesis algorithm that made the search algorithm inefficient [10].

3 Newly Implemented Search Algorithms

We now detail the search algorithms explored in this study.

```
do parallel
  Randomize genome
end do
for generation = 1 to number_of_generations
  do parallel
    Evaluate genome
    Select mate genome from a nearby processor
    Cross genome with mate genome
    Mutate new genome
  end do
end for
```

Fig. 2. A parallel GA.

```
for evaluation = 1 to number_of_evaluations
  Randomly generate a new genome
  Evaluate the new genome
  if the new genome is better than best_genome then
    Set best_genome = new genome
  end if
end for
```

Fig. 3. Random generate and test (RG&T).

3.1 Random Generate and Test

As a baseline benchmark, we implemented a simple *random generate-and-test* (RG&T) algorithm, which randomly generates a specified number of BSR controllers, evaluates their fitnesses, and selects the best controller generated (Figure 3).

3.2 Genetic Algorithms

Because of the success of the massively parallel GA, the first serial algorithms we implemented were genetic algorithms, distinguished from EP algorithms by the use of a crossover operator to recombine parameters among members in an evolving population of solutions.

3.2.1 Implementation of Genetic Operators

Both the crossover and mutation operators were tailored to fit the BSR representation.

```
Initialize population
for generation = 1 to number_of_generations
  Evaluate each genome in population
  for i = 1 to (size_of_population / 2)
    Select two parent genomes by roulette-wheel selection
    Cross & mutate to generate two child genomes
  end for
  Replace old parent population with new child population
end for
```

Fig. 4. A generational-replacement GA (GGA).

Crossover When two individuals are mated, two children are generated. One child, designated a hybrid, is a product of crossover between the two parents. The other is an exact copy of one of the parents (selected randomly).[3] Crossover between two genomes begins by creating a random 1:1 mapping between their stimulus-response pairs. Each stimulus-response pair is crossed with the one to which it is mapped, using a crossover operator tailored to this application [11].

Mutation Mutation perturbs a genome to which it is applied in two ways. One SR pair is subjected to *creep*, i.e., each of the parameters in that SR pair is changed by a small amount. Another SR pair is randomized from scratch, with the constraint that at least one corner of the new stimulus hyperrectangle coincide with the original trajectory through the multidimensional space defined by the stimulus senses.

3.2.2 Generational Replacement GA

The *generational-replacement genetic algorithm* (GGA) is characterized by the replacement of the entire population of genomes at each iteration. In our implementation (Figure 4), parent genomes are selected using roulette-wheel selection [9], in which the probability of a genome being selected for mating is proportional to its fitness. Two child genomes are produced from the parents. One child, randomly chosen, is the hybrid result of a crossover operation between the parents, and the other child is a copy of one parent. Both child genomes next undergo mutation. When the population of child genomes has been generated, it completely replaces the population of parent genomes, except for the best member of each generation, which is carried over to the next generation without modification by *elitism*. Of the algorithms presented in this study, this is the most similar to the massively parallel algorithm used by Ngo and Marks, although there are no analogues to the localized mating scheme and hill-climbing random initialization used in the parallel algorithm.

[3] This asymmetric mating strategy yielded better results than a symmetric strategy in which both offspring were hybrids.

```
Initialize population
Evaluate each genome in population
Rank order the population
for evaluation = 1 to (number_of_evaluations / 2)
  Select two parent genomes by linear rank-based selection
  Cross & mutate to generate two child genomes
  Evaluate the two child genomes
  Insert child genomes in order into population
  Delete two lowest-ranked genomes in the population
end for
```

Fig. 5. A steady-state GA (SSGA).

3.2.3 Steady-State GA

Unlike a GGA, a *steady-state genetic algorithm* (SSGA) uses an overlapping population in which only a small fraction of the population is replaced during each iteration. Previous researchers have reported that SSGAs outperform (in terms of speed on serial hardware) their generational counterparts in many applications [3]. The advantage of a SSGA is that newly generated individuals with high fitnesses are immediately available to take part in reproduction, rather than having to wait a complete generation until they become part of the mating pool.

The SSGA we used (Figure 5) uses a technique called *linear rank-based selection* [16]. The population is sorted according to fitness, and probabilities for being selected for reproduction are assigned based on the rank of the individual in the population, ensuring constant selective pressure throughout the search. A linear function is used to allocate reproductive trials, according to a user-defined *bias*. A bias of 1.5, for example, means that the top-ranked individual is 1.5 times more likely to reproduce than the median individual in a population. At each iteration, two parent genomes are selected according to this selection scheme and mated, producing two child genomes just as in the GGA. They are then inserted into the population according to their fitness. Thus, a maximum of two genomes will be replaced during each iteration.

3.2.4 Distributed GA

In a *distributed genetic algorithm* (DGA) [14], large populations are subdivided into smaller subpopulations or *demes*. A GA is executed independently for each deme, and the demes interact periodically by periodic migration of individuals among demes. This model maps naturally to a distributed workstation cluster, but also works well in single-machine serial architectures because there is very little overhead for subdividing a large population into demes. Tanese [14] has shown that DGAs implemented on serial machines outperform single population GAs, even in the absence of parallelism, and refers to this as the *superlinear*

```
Initialize all demes
Evaluate each genome in population
Rank order each deme
for evaluation = 1 to (number_of_evaluations / (2×number_of_demes))
  for each deme
    Insert any inbound migrants into deme
    Select two parent genomes by linear rank-based selection
    Cross & mutate to generate two child genomes
    Evaluate the two child genomes
    Insert child genomes in order into population
    Delete two lowest-ranked genomes in the population
    With small probability
      Select two migrant genomes by linear rank-based selection
      Send copies of migrant genomes to randomly selected deme
  end for
end for
```

Fig. 6. A distributed GA (DGA).

speedup phenomenon of distributed GAs. The improved performance of a DGA is believed to be due to niching caused by reproductive isolation between demes.

In our DGA (Figure 6), a SSGA identical to the one described above in Section 3.2.3 is executed for each deme. A migration operator is used with small probability to establish gene flow among the isolated demes.

3.3 Evolutionary Programming Algorithms

3.3.1 Evolutionary Programming

Evolutionary programming (EP) [4, 5] is a class of evolutionary computation algorithms that can be distinguished from genetic algorithms primarily by the lack of crossover and other genetic operators. EP1 (Figure 7) and EP2 (Figure 8) are EP algorithms that are based directly on our previously described generational and steady-state genetic algorithms, respectively. They differ from their GA counterparts only in that the crossover operator is never applied, while the mutation operator is applied with probability 1.

3.3.2 Stochastic Hill Climbing

Stochastic hill climbing (SHC) is the simplest possible EP algorithm (Figure 9). A single initial solution is perturbed randomly at each iteration, using the mutation operator used in the genetic algorithms (but with a probability of 1 that the operator is applied). The resulting child genome is evaluated and compared to the original genome, and the better

```
Initialize population
for generation = 1 to number_of_generations
  Evaluate each genome in population
  for i = 1 to size_of_population
    Select a parent genome by roulette-wheel selection
    Mutate to generate a child genome
  end for
  Replace old parent population with new child population
end for
```

Fig. 7. Evolutionary programming (EP1).

```
Initialize population
Evaluate each genome in population
Rank order the population
for evaluation = 1 to number_of_evaluations
  Select a parent genome by linear rank-based selection
  Mutate to generate a child genome
  Evaluate the child genome
  Insert child genome in order into population
  Delete the lowest-ranked genome in the population
end for
```

Fig. 8. Evolutionary programming (EP2).

```
Initialize and evaluate a single genome
for evaluation = 1 to number_of_evaluations
  Randomly perturb the genome
  Evaluate the new genome
  if the new genome is better than the old one then
    Replace the old genome with the new one
  end if
end for
```

Fig. 9. Stochastic hill climbing (SHC).

genome is selected as the parent genome for the next iteration. A likely problem with this simple algorithm is that it is easy for the search to be trapped at a local optimum.

3.3.3 Stochastic Population Hill Climbing

The *stochastic population hill climbing* (SPHC) algorithm (Figure 10) improves on the SHC algorithm by using a population of solutions to add robustness to the search. However, rather than applying selection to the population at every iteration, as is commonly done with an EP algorithm, each member independently undergoes stochastic hill climbing. Periodically, a *reseeding* operator is applied which selects the top half of the population and copies them into the bottom half of the population, refocusing the search on the most promising genomes in the population.

4 Experimental Study

A study was designed to evaluate the performance of the newly implemented serial algorithms. For animation applications, the visual quality and physical realism of the trajectories found by the searches is the ultimate measure of performance. However, these subjective metrics are not easily obtainable. We therefore used the fitness values calculated by the evaluation function as an objective measure of success.

Each of the serial algorithms described in Section 3 was tested on five different instances of the SC problem (see the appendix for a description of these problems). Each algorithm was run until the evaluation function was executed 40,000 times. For example, in the case of the GGA, this means that if the population size was 100, then 400 generations were executed. On the other hand, for a SHC, the initial individual was subjected to 40,000 random perturbations. The number 40,000 was chosen based on early trials that indicated that controllers that generated high-quality motions could be generated within 40,000 evaluations.

Based on performance on early, small-scale experiments, each algorithm was tested

```
Initialize population
Evaluate each genome in population
for generation = 1 to number_of_generations
  for each individual genome in the population
    Randomly perturb the genome
    Evaluate the new genome
    if the new genome is better than the old one then
      Replace the old genome with the new one
  end for
  if (generation mod reseed_interval) = 0 then
    Rank order the population
    Replace bottom 50% of the population with top 50%
  end if
end for
```

Fig. 10. Stochastic population hill climbing (SPHC).

using several sets of promising control parameters (Table 4). In Table 4, the crossover and mutation rates refer to the probability that the operators were applied to a given genome. The migration operator is the probability that outbound migrants were generated out of a particular deme. The *reseed interval* is the number of generations between reseeding operations in the SPHC algorithm, where a generation is a number of evaluations equal to the size of the population.

For each experimental group (an algorithm + control parameter set), each experiment was repeated 10 times, and the average performance was calculated.

5 Results

5.1 Comparison of Serial Algorithms

Figures 11 through 15 present the performances (as measured by the fitness values of the best individuals evaluated over time) of a representative group from each algorithm for each of the five SC problems. The groups selected to be shown are the groups that performed the best overall in all five of the test-suite problems.

From the figures, the following general observations can be made:

- The EP algorithms outperformed everything else. In particular, the SPHC algorithm consistently yielded the best results.
- All the EC algorithms outperformed RG&T.
- The SHC algorithm performed very well in the beginning, but its progress tapered off, and was consistently outperformed by the SPHC algorithm.

Experimental Group	*Population*	*Mutation Rate*	*Crossover Rate*	*Migration Rate*	*Reseed Interval*
RG&T	N/A	N/A	N/A	N/A	N/A
GGA-1	100	0.1	0.6	N/A	N/A
GGA-2	200	0.1	0.6	N/A	N/A
SSGA-1	100	0.1	0.6	N/A	N/A
SSGA-2	200	0.1	0.6	N/A	N/A
DGA-1	5×40	0.1	0.6	0.05	N/A
DGA-2	5×40	0.1	0.6	0.005	N/A
DGA-3	5×100	0.1	0.6	0.05	N/A
DGA-4	5×100	0.1	0.6	0.005	N/A
EP1-1	20	1.0	N/A	N/A	N/A
EP1-2	50	1.0	N/A	N/A	N/A
EP2-1	20	1.0	N/A	N/A	N/A
EP2-2	50	1.0	N/A	N/A	N/A
SHC	1	1.0	N/A	N/A	N/A
SPHC-1	10	1.0	N/A	N/A	100
SPHC-2	10	1.0	N/A	N/A	200
SPHC-3	10	1.0	N/A	N/A	400
SPHC-4	10	1.0	N/A	N/A	400
SPHC-5	10	1.0	N/A	N/A	800

Table 2. Key parameters.

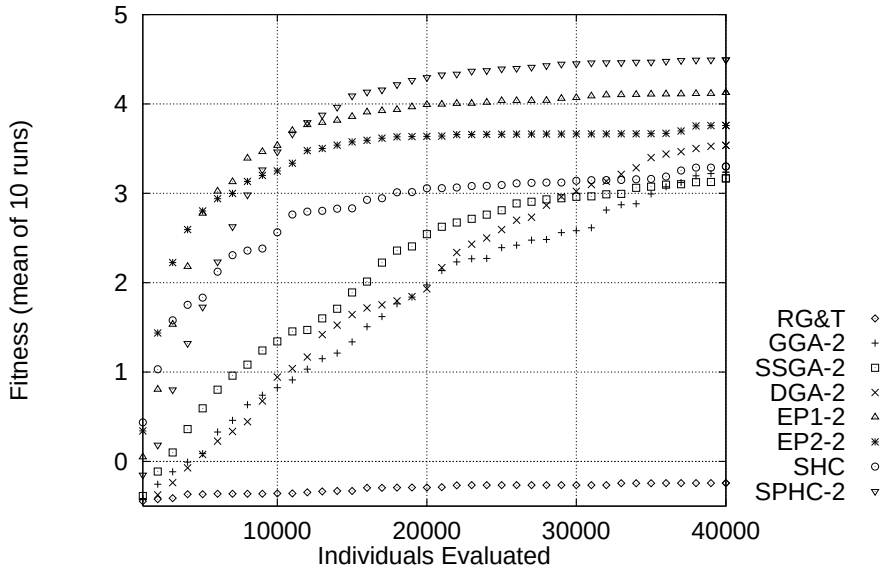

Figure 11: Comparative performance of the algorithms, Sarah Sigma Problem (n=10)

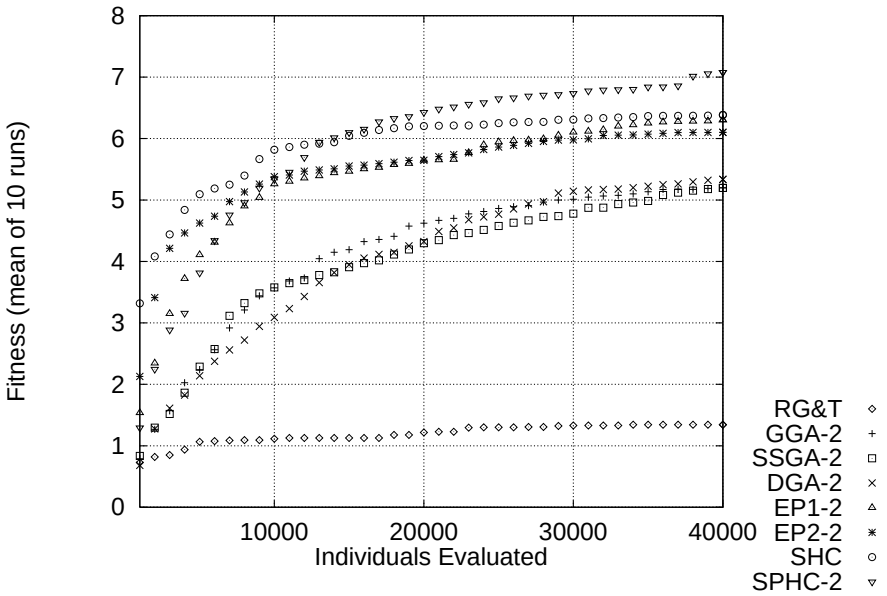

Figure 12: Comparative performance of the algorithms, Willy Worm Problem (n=10)

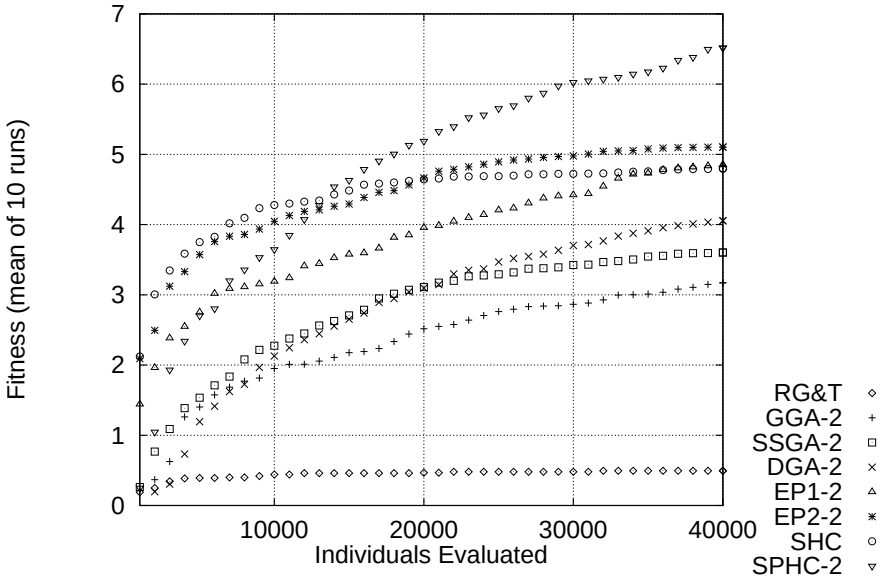

Figure 13: Comparative performance of the algorithms, Beryl Biped Problem (n=10)

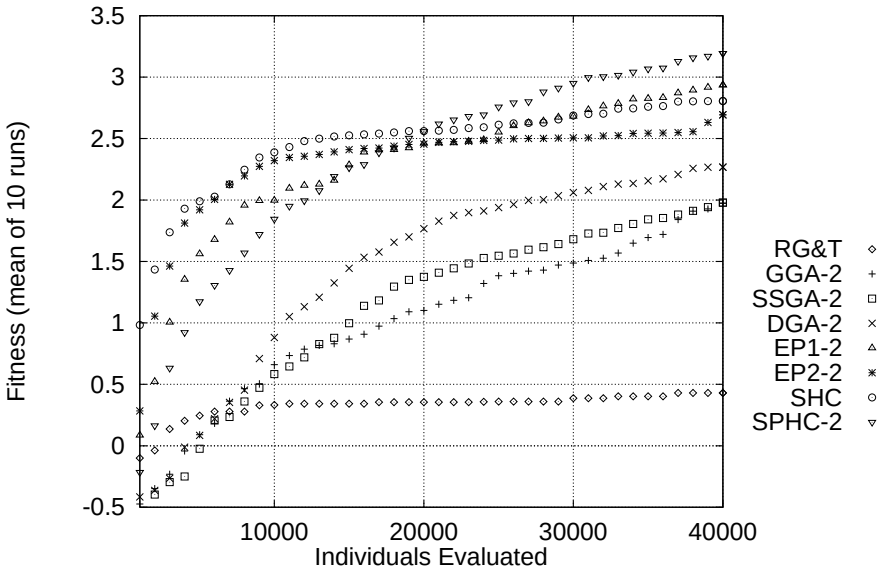

Figure 14: Comparative performance of the algorithms, Mr. Star-Man Problem (n=10)

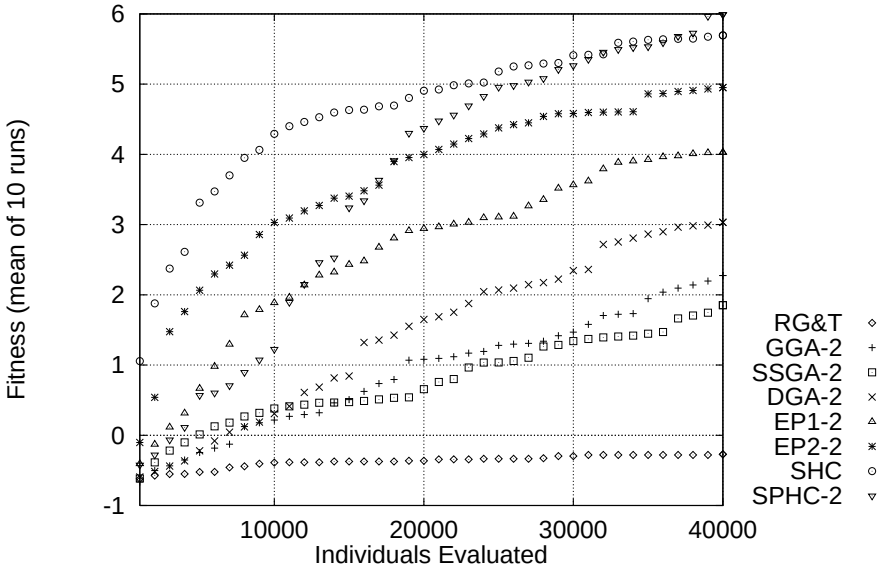

Figure 15: Comparative performance of the algorithms, Five-Rod Fred Problem (n=10)

- There is no consistent trend in performances among the genetic algorithms (i.e. GGA, SSGA, DGA).

An additional observation is that the less successful EC algorithms were eventually able to generate solutions of the same quality as the SPHC algorithm if we let them proceed beyond 40,000 evaluations. That is, the primary difference observed between the algorithms was their efficiency, and not in their completeness (ability to generate good solutions).

5.2 Comparisons with the Massively Parallel GA

Due to the enormous computational cost that would have been incurred on a CM-2, we did not obtain performance data for the massively parallel GA that could be compared quantitatively with the serial algorithms. However, we can make the following anecdotal observations about the relative performance of the serial and parallel algorithms:

- The parallel GA requires between 200,000 and 850,000 physical simulations (50-200 generations of the GA) to produce motion controllers comparable to those produced at a cost of 40,000 simulations by the SPHC algorithm.
- The running time of the massively parallel GA is typically 30-60 minutes on a 4096-processor CM-2 Connection Machine, versus 3-6 minutes for the 40,000 evaluation SPHC algorithm on a DEC 3000/400 AXP workstation.

We also observed that, in the case of SC problems with highly multimodal solutions, our EC algorithms are able to generate all of the major variations that the massively parallel GA was able to generate. For example, for the Willy Worm locomotion problem, the SPHC algorithm generates both the "crawling" and "flipping" modes of locomotion (Figures 18-19).

6 Experiments with Standard Continuous Optimization Methods

In addition to the stochastic discrete optimization methods outlined above, we also experimented with more traditional continuous optimization methods. We implemented two techniques commonly applied in continuous optimization problems lacking gradient information: the downhill simplex method of Nelder and Mead and the direction-set technique of Powell. Complete implementation details of these algorithms are beyond the scope of this paper; however, descriptions and sample implementations can be found in [13].

6.1 Powell's Method

Powell's method is classified as a "direction-set" optimization algorithm, which is one that seeks to calculate an optimal basis for the function space such that the unit vectors are well suited to one-dimensional optimization. In other words, an optimal choice of basis

vectors should provide steep descent, while at the same time satisfying the criterion that optimization along one dimension disturbs minimally previously computed optimizations along other basis dimensions. Powell's algorithm, then, works by repeatedly performing successive one-dimensional optimizations along a set of basis vectors. As it progresses, the orientation of the set of basis vectors is revised to increase the rate of descent. Our implementation utilized Brent's method for one-dimensional optimization, though other techniques would probably work just as well given the lack of differentiability exhibited by much of the search space.

6.2 The Downhill Simplex Method of Nelder and Mead

This approach works by keeping a simplex of points and at each iteration repositioning the highest point of the simplex by reflecting it through the opposite face. If this is unsuccessful, the algorithm tries to squeeze the simplex along an axis, or, if nothing else works, to shrink the simplex. The algorithm continues in this fashion, moving the simplex downhill until it is no longer able to make progress. The main advantage of this algorithm, other than simplicity, is that it makes no assumptions about the smoothness or differentiability of the space. Also, because of its coarseness, it can sometimes "step over" local minima that trap Powell's method. Because of the discontinuous nature of the BSR search space, we expected that this method might prove to be more robust than Powell's method, which relies more heavily on assumptions of smoothness.

6.3 Experiments and Results

We performed two groups of experiments to determine the effectiveness of these techniques for motion synthesis. In the first group of tests, we applied the algorithms from scratch, generating a random point in BSR space and using it to seed the minimization algorithms. In the second group of tests, we considered using these techniques as a postprocess to refine existing trajectories. For these experiments, we began with an existing solution generated by the SPHC algorithm and then ran the minimization algorithms on this solution to determine if further improvement of the solution was obtainable. Each set of experiments was performed for seven distinct motion/creature combinations.

In the first group of experiments, we found that neither of the minimization techniques proved to be effective at generating trajectories from scratch. For each of the seven motion-synthesis problems, a hundred random seeds were generated for each technique. Since it is common to restart multidimensional minimization techniques such as these, each algorithm was restarted five times with the previous solution. (In most cases no additional improvement was obtained after one or two restarts — five was chosen as a reasonably secure upper bound.) Nevertheless, neither of the techniques was able to provide a useful amount of improvement beyond the initial randomly generated solutions, for any of the seven test cases. Although experiments with a larger number of random seeds could be considered, we found that our computational cost for the above experiments was already greater than the corresponding cost required by the stochastic search techniques described earlier to find near optimal solutions.

Figure 16: Improvements to BSR controllers using Powell's method and the downhill simplex method

In the second groups of tests, we applied each of these techniques to a solution discovered by the SPHC algorithm after 40,000 iterations. In nearly all cases, these techniques provided at least a modest amount of improvement (see Figure 16). As the graph shows, in nearly all cases Powell's method provides at least as much improvement as the downhill simplex method.

Overall, our experiments indicate that neither of these techniques is likely to be useful for generating interesting motions from scratch. Nevertheless, our results indicate that it may be useful to consider the use of continuous optimization algorithms such as these for local refinement of trajectories generated by discrete search techniques. Not surprisingly, Powell's method is often able to fine-tune solutions found by the SPHC algorithm to a degree which SPHC itself is unlikely to achieve in a reasonable amount of time, due to the coarseness and randomness of its mutation operators. The visual effect of this improvement, however, is often quite modest.

7 Conclusions

The primary motivation for this study was to find a search algorithm that would make the Ngo-Marks approach to motion synthesis viable on current serial machines. We have succeeded in this effort, and have identified the SPHC algorithm to be a particularly effective serial global search algorithm. We have improved the efficiency of the global search algorithm by an order of magnitude with respect to the number of physical simulations executed; furthermore, the SPHC algorithm algorithm on a DEC AXP workstation generates BSR controllers equivalent to those found by the massively parallel GA in one tenth of the time.

Another significant result yielded by our study was the isolation of the BSR controller representation as *the* key component in the success of the motion-synthesis algorithm. The

fact that simple search algorithms such as SPHC are successful in generating good motion controllers, even outperforming the original massively parallel GA, clearly demonstrates the fact that the space of BSR controllers is relatively simple to search for near-optimal solutions.

We believe that our study is thorough enough to conjecture that although there no doubt exist other algorithms that can marginally outperform the SPHC algorithm, future research in improving the efficiency of the the search component of the motion-synthesis algorithm would be best directed towards studying fundamentally different search algorithms, different encodings of the motion controller, or substantially different implementations of genetic operators. For example, our preliminary experiments with two standard function-optimizing techniques has shown that although these deterministic algorithms perform poorly from random starting points, they can rapidly find small to medium improvements in solutions originally generated by our EC algorithms (Section 6). A hybrid EC/deterministic algorithm may prove to be a very effective combination.

References

[1] L. S. Brotman and A. N. Netravali. Motion interpolation by optimal control. *Computer Graphics*, 22(4):309–315, August 1988.

[2] M. F. Cohen. Interactive spacetime control for animation. *Computer Graphics*, 26(2):293–302, July 1992.

[3] L. Davis. *Handbook of Genetic Algorithms.* Van Nostrand Reinhold, New York, NY, 1991.

[4] Evolutionary Programming Society. *Proceedings of the First Annual Conference on Evolutionary Programming*, February 1992.

[5] Evolutionary Programming Society. *Proceedings of the Second Annual Conference on Evolutionary Programming*, February 1993.

[6] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes. *Computer Graphics: Principles and Practice.* The Systems Programming Series. Addison-Wesley, Reading, MA, 2nd edition, 1990.

[7] A. Fukunaga. Genetic and stochastic search strategies to solve the spacetime constraints problem. A.B. Thesis, Harvard University, April 1993.

[8] A. Fukunaga, J. T. Ngo, and J. Marks. Automatic control of physically realistic animated figures using evolutionary programming. In *Proceedings of the Third Annual Conference on Evolutionary Programming (EP94)*, San Diego, CA, February 1994. World Scientific. To appear.

[9] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, Reading, Massachusetts, 1989.

[10] J. T. Ngo and J. Marks. Massively parallel genetic algorithm for physically correct articulated figure locomotion. Working Notes for the AAAI Spring Symposium on Innovative Applications of Massive Parallelism, Stanford University, March 1993.

[11] J. T. Ngo and J. Marks. Physically realistic motion synthesis in animation. *Evolutionary Computation*, 1(3):235–268, 1993.

[12] J. T. Ngo and J. Marks. Spacetime constraints revisited. In *SIGGRAPH '93 Conference Proceedings*, pages 343–350. ACM SIGGRAPH, Anaheim, CA, August 1993.

[13] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C. The Art of Scientific Computing.* Cambridge University Press, Cambridge, UK, second edition, 1992.

[14] R. Tanese. Distributed genetic algorithms. In *Proceedings of the Third International Conference on Genetic Algorithms*, 1989.

[15] M. van de Panne and E. Fiume. Sensor-actuator networks. In *SIGGRAPH '93 Conference Proceedings*, pages 335–342, Anaheim, CA, August 1993. ACM SIGGRAPH.

[16] D. Whitley. The genitor algorithm and selection pressure: Why rank-based allocation of reproductive trials is best. In *Proceedings of the Third International Conference on Genetic Algorithms*, 1989.

[17] A. Witkin and M. Kass. Spacetime constraints. *Computer Graphics*, 22(4):159–168, August 1988.

A Test Suite of Spacetime Constraint Problems

A.1 Sarah Sigma

Sarah Sigma (Figure 17) is an unbranched, four-rod creature whose task was to jump as high as possible. The evaluation criterion was the maximum height achieved by the lowest joint during a period of 50 time steps. The joint angle ranges on Sarah are quite limited, so the motions that result are predictable. There is an initial compress phase (*squash*), followed by a rapid expansion phase (*expand*), which propels Sarah into the air. Once in the air, Sarah compresses again (*compress*), increasing the height of its lowest joint.

A.2 Willy Worm

Willy Worm (Figures 18-19) is an asymmetric, unbranched, three-rod creature with a large degree of freedom in its joint angle movement. The task given Willy is forward locomotion (walk as far as possible) within the time allotted. The evaluation criterion was the maximum horizontal distance achieved by the center of mass. This is an example of a multi-modal problem in which the various solutions are entirely distinct. There are two motions which are commonly observed (both shown here): *flipping* and *shuffling*.

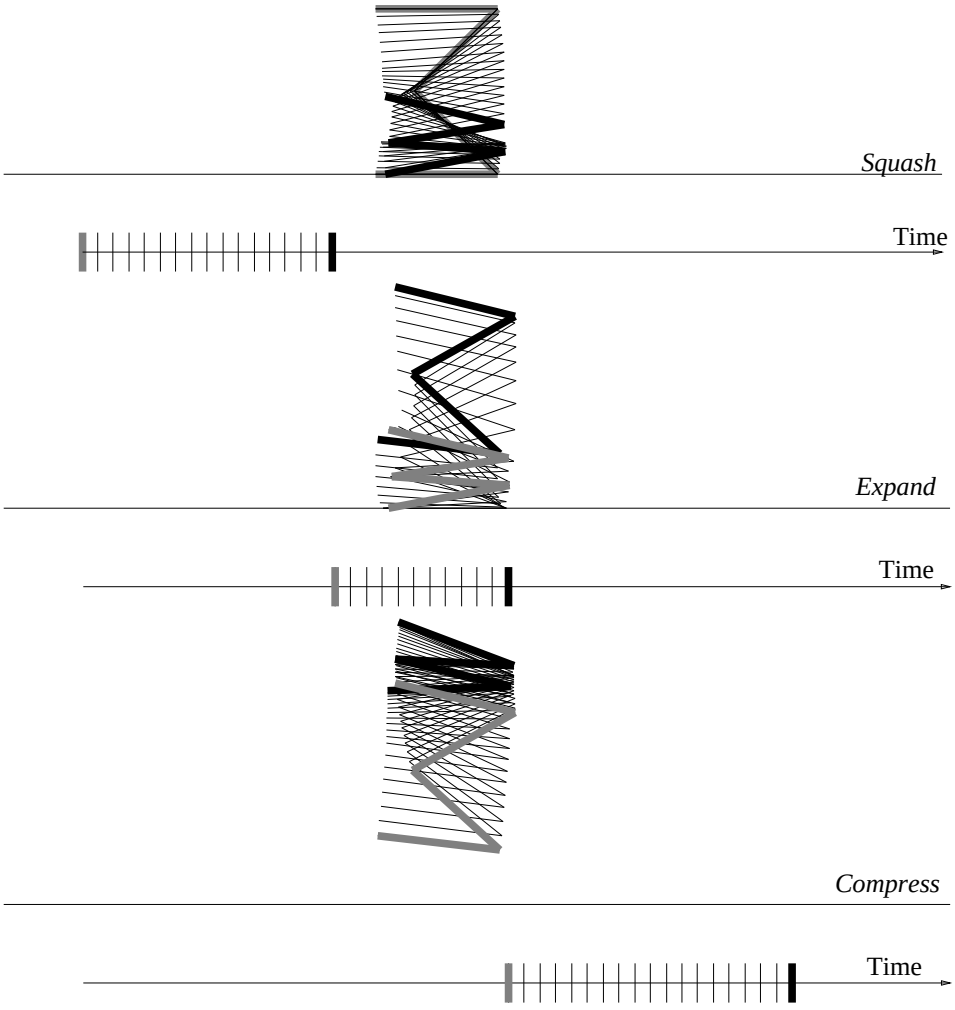

Figure 17: Sarah Sigma jumping.

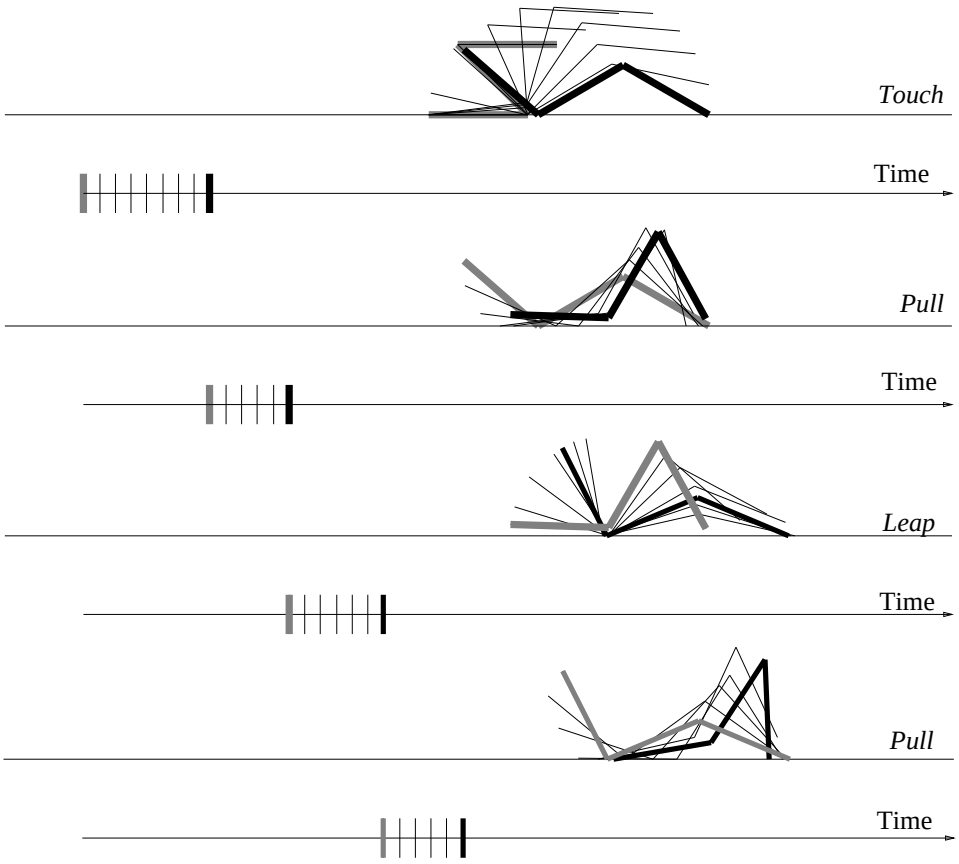

Figure 18: Willy Worm shuffling.

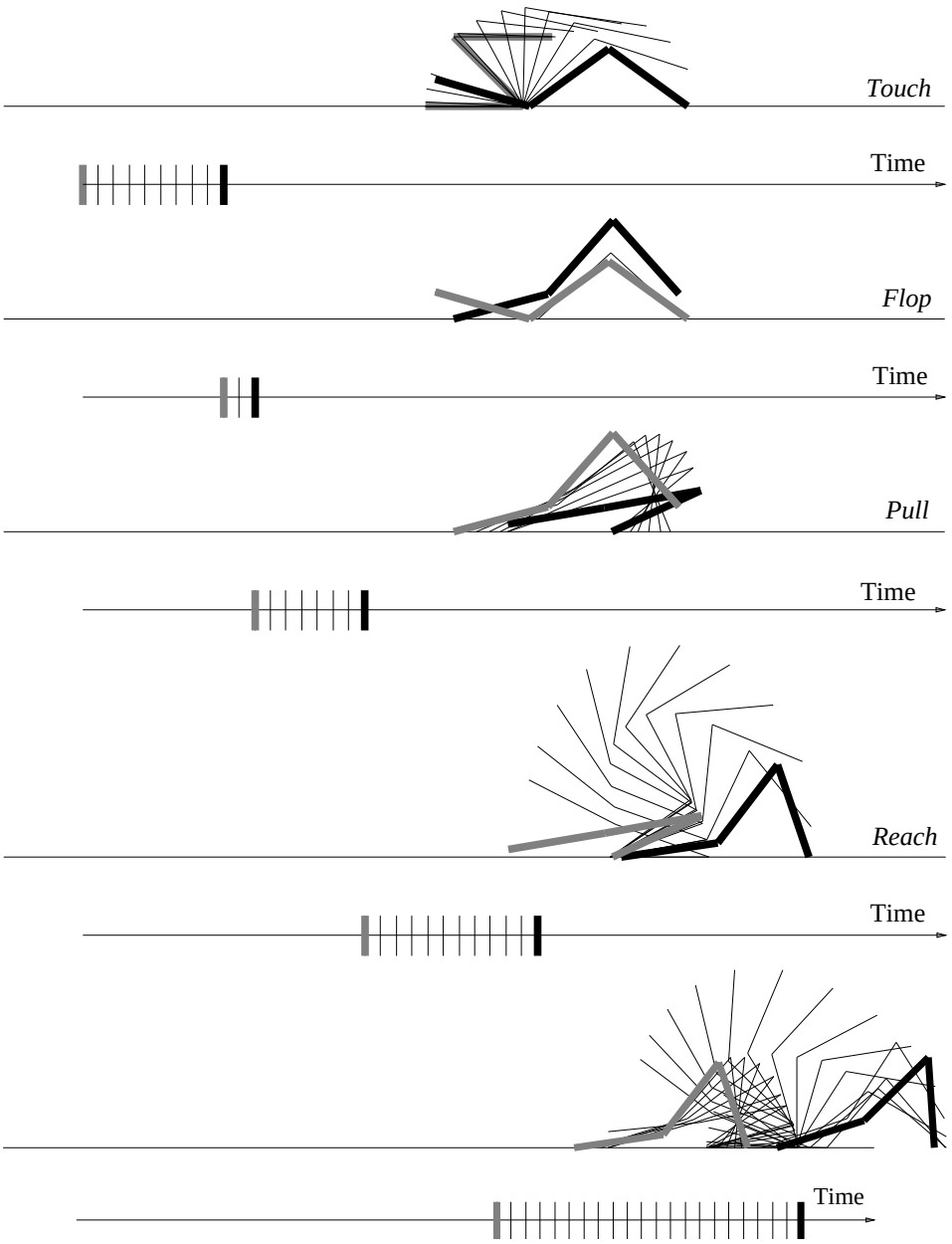

Figure 19: Willy Worm flipping forward.

A.3 Beryl Biped

Beryl Biped (Figure 20) is a branched, five-rod character with two jointed legs and a rigid torso, roughly modelling a two-dimensional legged humanoid whose task is forward locomotion. Rod masses are of human proportion. Beryl's success was measured by the maximum horizontal distance achieved by the center of mass during the course of a simulation. The resulting motion shown here is a cyclic, bipedal locomotion.

A.4 Mr. Star-Man

Mr. Star-Man (Figures 21-22) is a branched, five-rod creature, in which all rods are of equal length and mass. Joint-angle ranges are confined to one of the quadrants defined relative to its top "torso" rod, so that rods cannot cross each other. Given the task of forward locomotion,the two main classes of resulting motions (both shown here) are *cartwheeling* and *shuffling*.

A.5 Five-Rod Fred

Five-Rod Fred (Figures 23-24) is an unbranched creature consisting of five equal-length rods. The middle rods are of equal mass, but the terminal rods are five times heavier. Each joint allows its pair of connected rods to be at most 30 degrees from collinear. In addition to an inchworm-like crawling motion which was anticipated when given the task of forward locomotion, this creature yielded some of the most interesting motions developed by the system, including a flipping motion which results in forward momentum, culminating with a roll at the end.

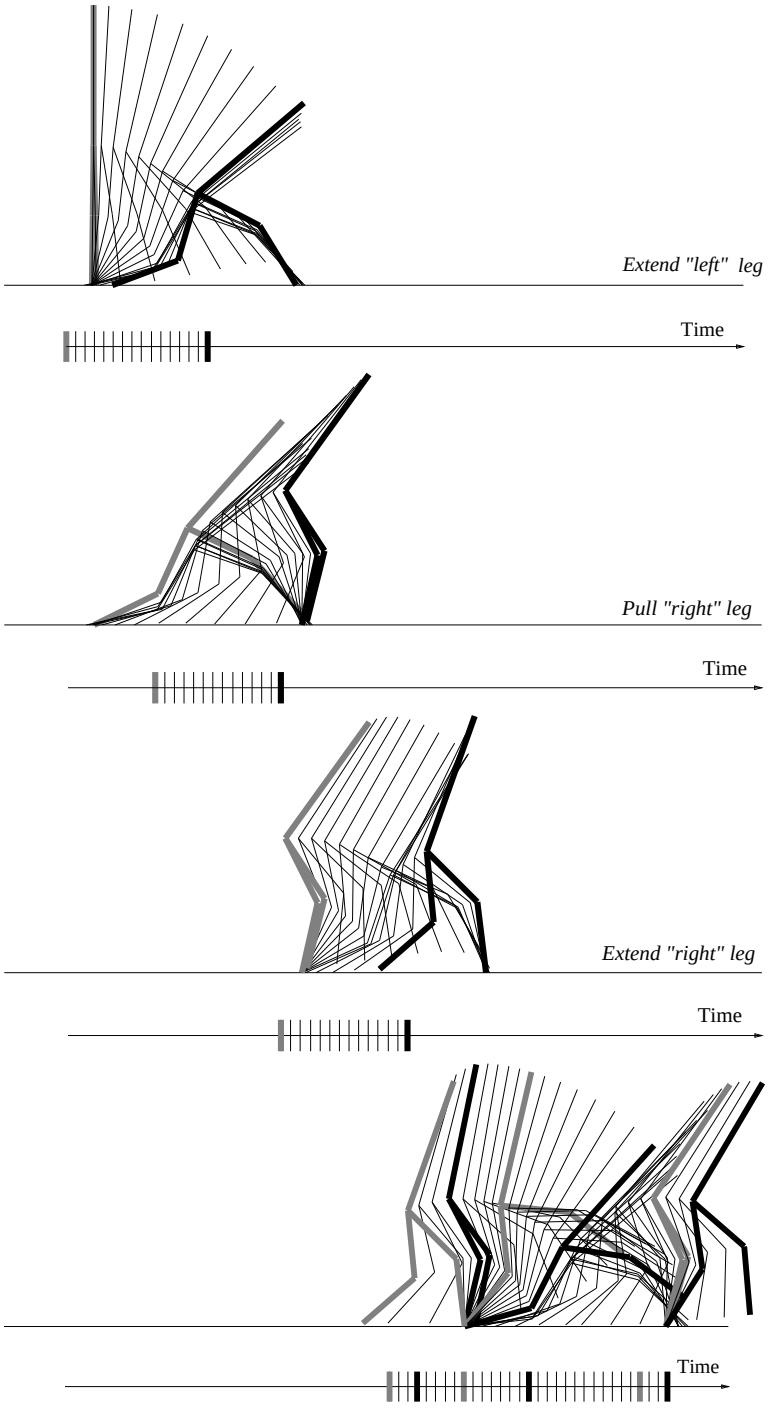

Figure 20: Beryl Biped walking.

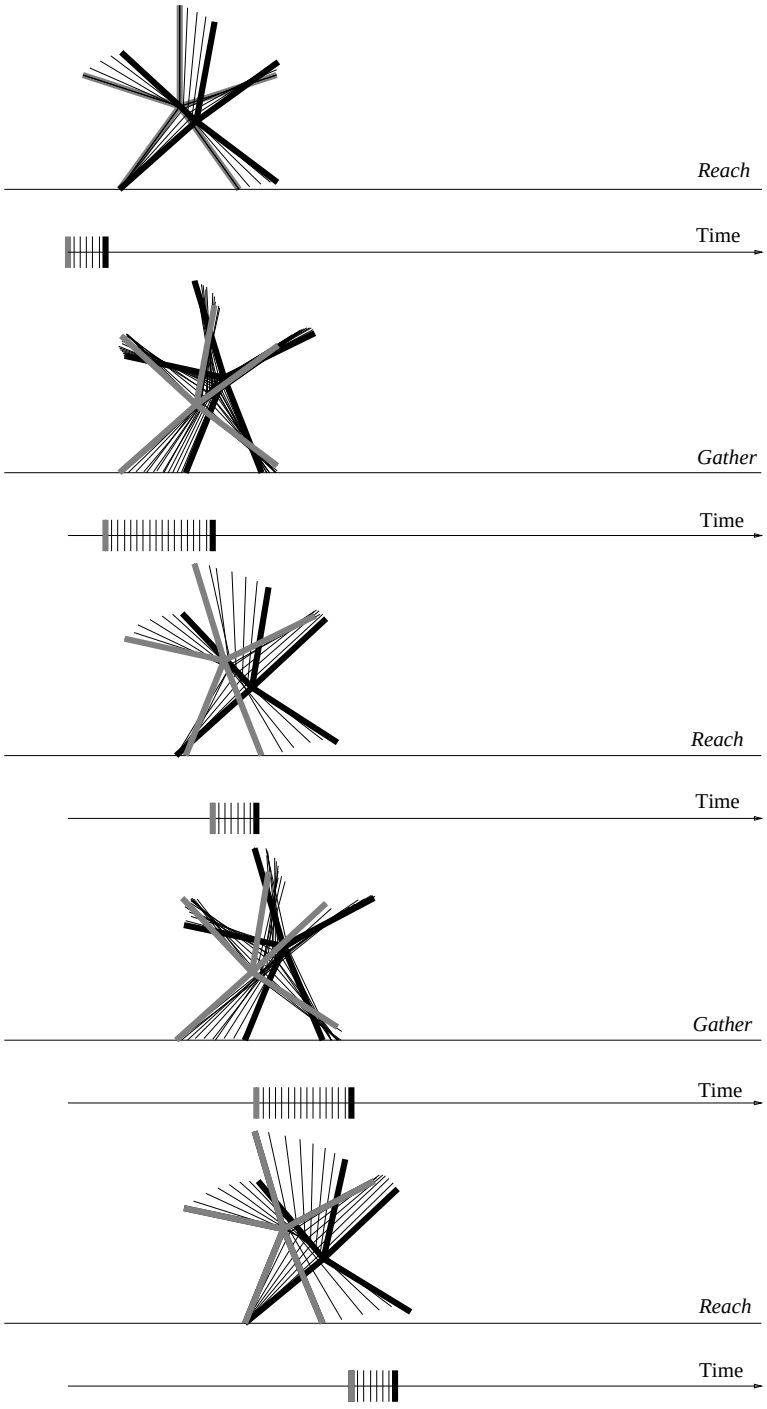

Figure 21: Mr. Star-Man shuffling.

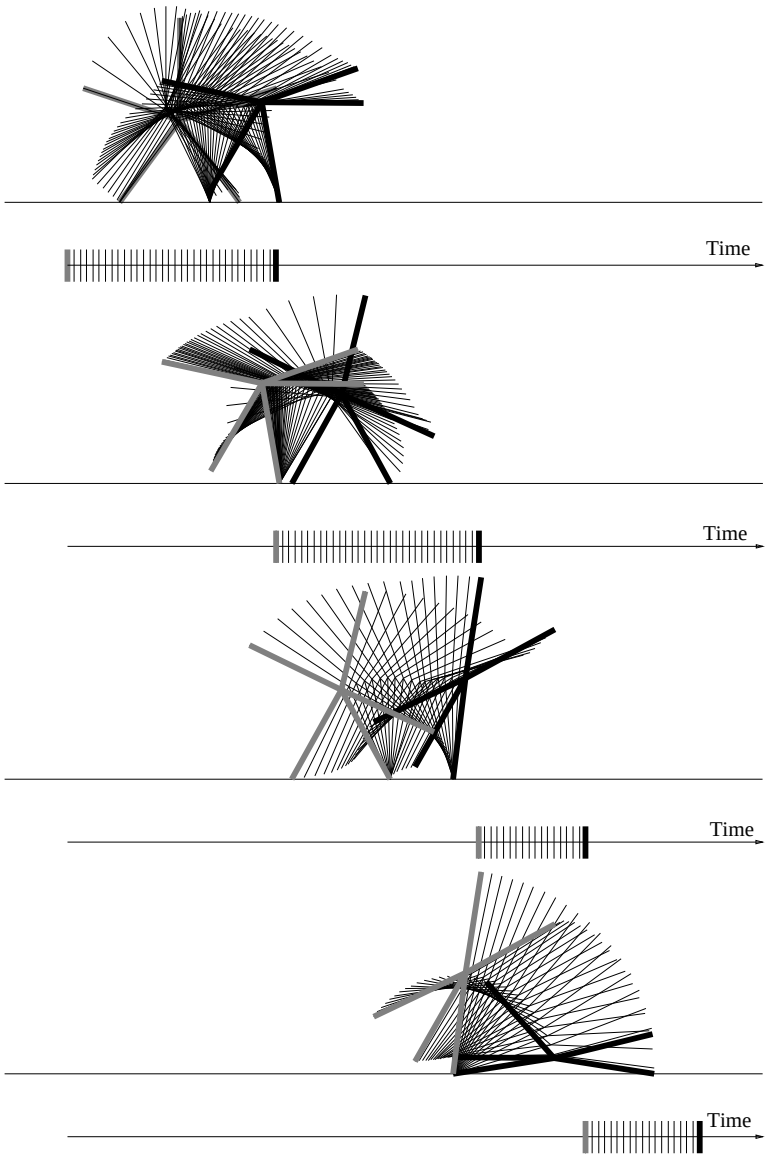

Figure 22: Mr. Star-Man cartwheeling.

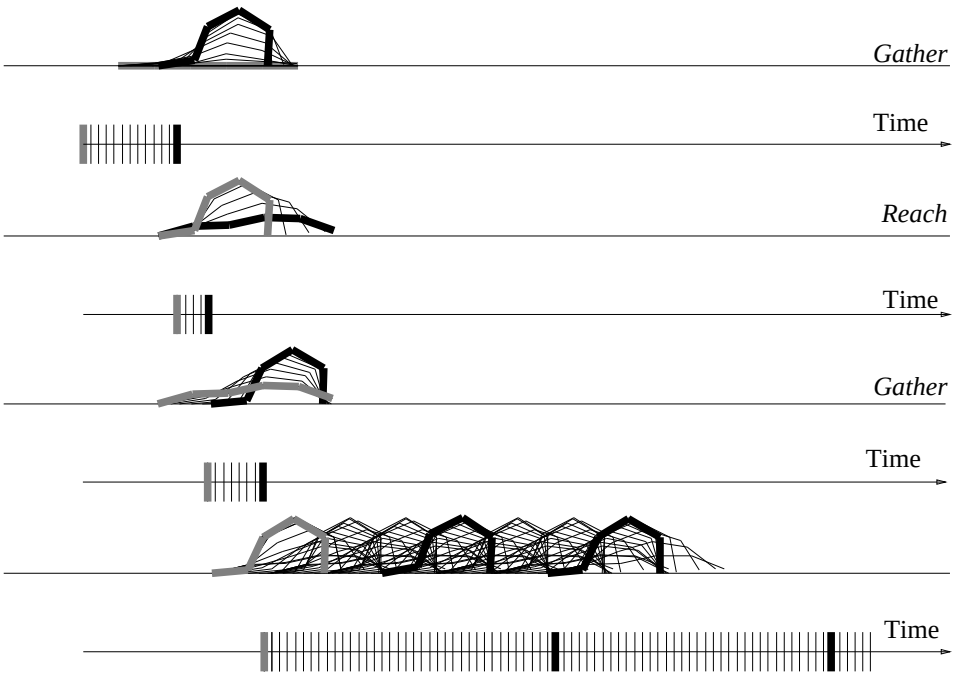

Figure 23: Five-Rod Fred crawling.

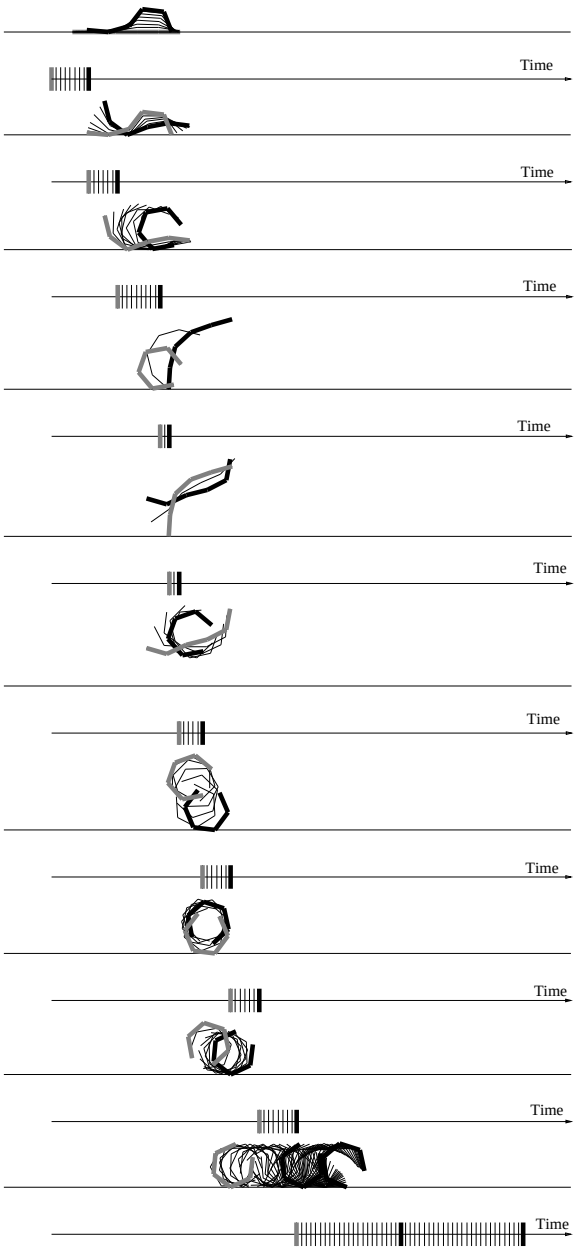

Figure 24: Five-Rod Fred flipping and rolling.